Problem Set #1: BNF

Thomas Moskal

CSC 344 – Professor Graci

Due date: 2/11/22

Learning Abstract: In this problem set assignment I will be tackling several exercises that center around BNF. I will be creating grammar descriptions and parse trees. There are many different languages to look at in this lesson. Some are more efficient and useful than others. Examining the rules that make up these languages will show the logical structure beneath them.

1 – 1)

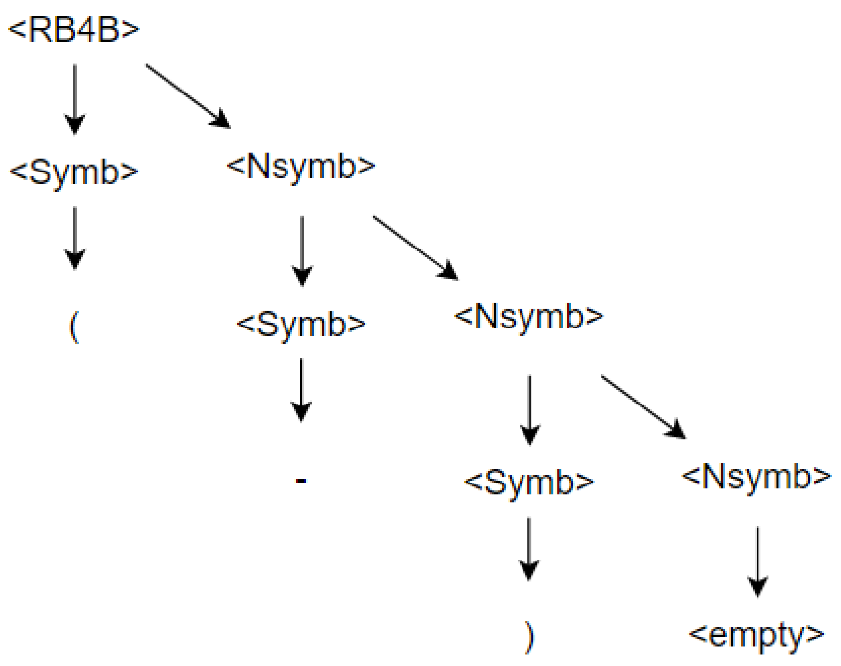
<RB4B> ::= <empty> | <Symb> <Nsymb>

<Symb> ::= (|) | - | . | + | <empty>

<Nsymb> ::= <Empty> | <Symb> | <Symb> <Nsymb>

1 – 2)

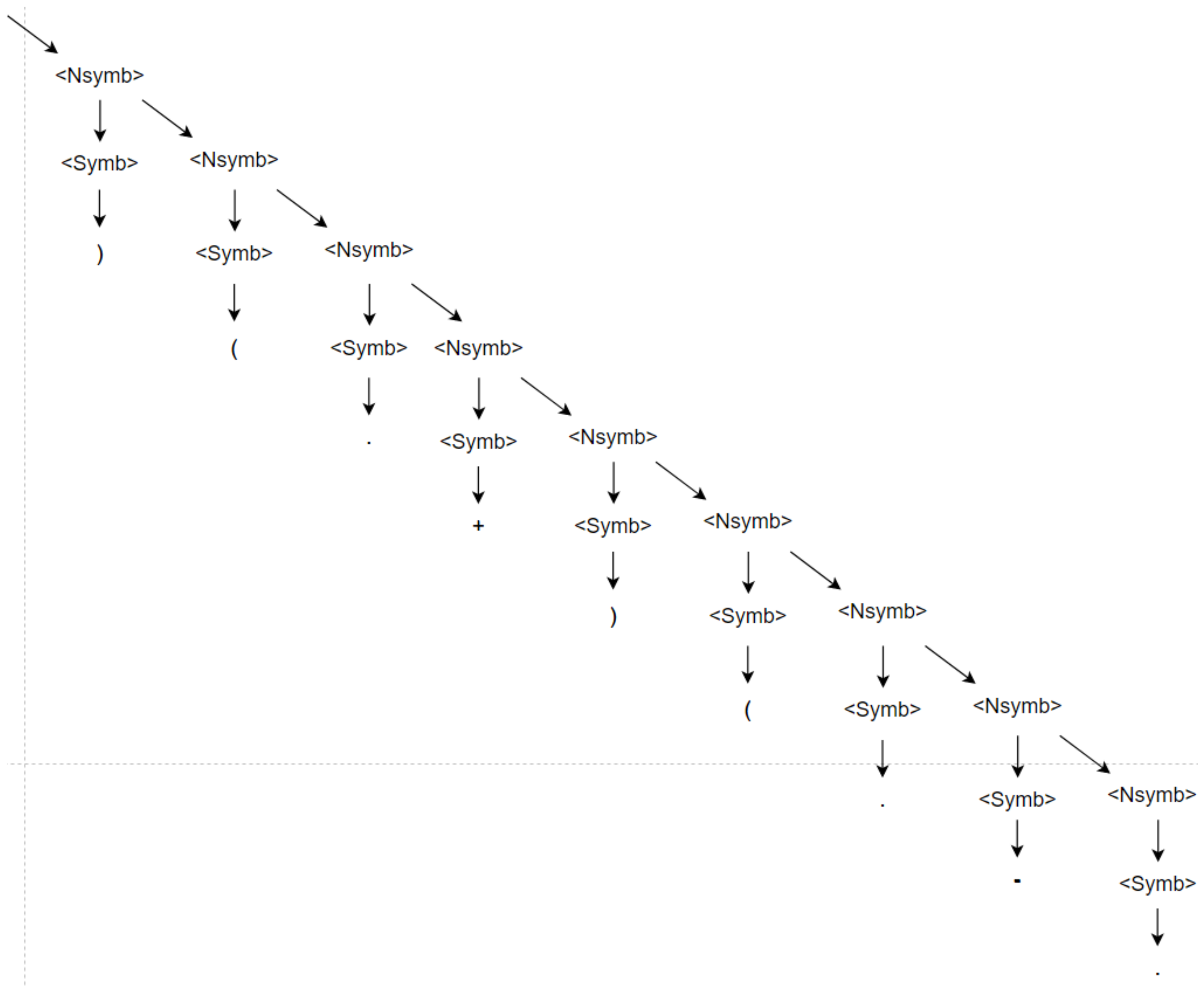
RB4B Parse Tree for: (-)

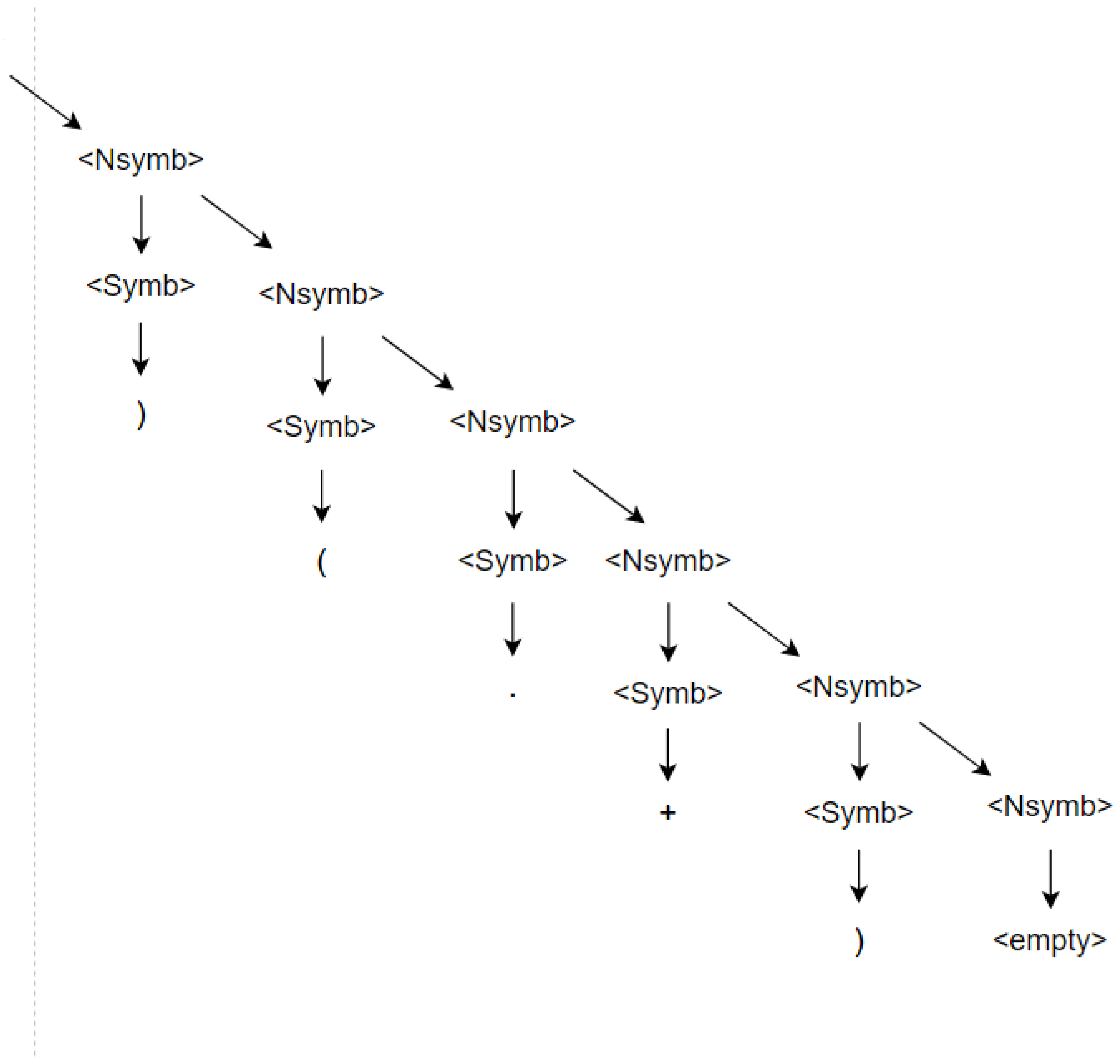


1 – 3)

RB4B Parse Tree for: (. . -)(. . -)(. +)(. - .)(. +)

$$\begin{tikzcd}
\langle\text{RB4B}\rangle \arrow[d] \arrow[dr] \\
\langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \arrow[dr] \\
(& \langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \arrow[dr] \\
 & . & \langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \arrow[dr] \\
 & & . & \langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \arrow[dr] \\
 & & & \text{-} & \langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \arrow[dr] \\
 & & & &) & \langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \arrow[dr] \\
 & & & & & (& \langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \arrow[dr] \\
 & & & & & & . & \langle\text{Symb}\rangle \arrow[d] & \langle\text{Nsymb}\rangle \arrow[d] \\
 & & & & & & & . & \langle\text{Symb}\rangle \arrow[d] \\
 & & & & & & & & \text{-}
\end{tikzcd}$$


<Nsymb>
<Symb>
)
<Nsymb>
<Symb>
(
<Nsymb>
<Symb>
.
<Nsymb>
<Symb>
+
<Nsymb>
<Symb>
)
<Nsymb>
<Symb>
(
<Nsymb>
<Symb>
.
<Nsymb>
<Symb>
-
<Nsymb>
<Symb>
.



Problem 2 – SQN (Special Suaternary Numbers)

2 – 1)

<SQN> ::= <num>

<num> ::=   0 <~num_0> | 1 <~num_1> | 2 <~num_2> | 3 <~num_3>

<~num_0> ::= <Empty> | 1 <~num_1> | 2 <~num_2> | 3 <~num_3>

<~num_1> ::= <Empty> | 0 <~num_0> | 2 <~num_2> | 3 <~num_3>

<~num_2> ::= <Empty> | 1 <~num_1> | 0 <~num_0> | 3 <~num_3>

<~num_3> ::= <Empty> | 1 <~num_1> | 2 <~num_2> | 0 <~num_0>

2 – 2)

SQN Parse Tree for: 0

$$\begin{tikzcd}
\langle \text{SQN} \rangle \arrow[d] & \\
\langle \text{num} \rangle \arrow[d] \arrow[dr] & \\
0 & \langle \text{\textasciitilde num\_0} \rangle \arrow[d] \\
 & \langle \text{empty} \rangle
\end{tikzcd}$$

2 – 3)

SQN Parse Tree for: 132

$$\begin{tikzcd}
\langle \text{SQN} \rangle \arrow[d] & & & \\
\langle \text{num} \rangle \arrow[d] \arrow[dr] & & & \\
1 & \langle \text{\textasciitilde num\_1} \rangle \arrow[d] \arrow[dr] & & \\
 & 3 & \langle \text{\textasciitilde num\_3} \rangle \arrow[d] \arrow[dr] & \\
 & & 2 & \langle \text{\textasciitilde num\_2} \rangle \arrow[d] \\
 & & & \langle \text{empty} \rangle
\end{tikzcd}$$

2 – 4)

I cannot draw a parse tree consistent with my grammar for this language for 1223 for one simple reason. I have no rule in my grammar structure that allows for a 2 to be placed before or after another 2. Logically, it is impossible.

Problem 3 – IR123

3 – 1)

<IR123> ::= <cell>

< cell > ::= [C] <~ cell[C]> | [D C] <~ cell[DC] > | [B C] <~ cell[BC] > | [E D C] <~ cell[EDC]> | [F E C] <~ cell[FEC] > | [G F C] <~ cell[GFC] >

<~ cell[C]> ::= <Empty> | [D C] <~ cell[DC] > | [B C] <~ cell[BC] > | [E D C] <~ cell[EDC]> | [F E C] <~ cell[FEC] > | [G F C] <~ cell[GFC] >

<~ cell[DC] > ::= <Empty> | [C] <~ cell[C]> | [B C] <~ cell[BC] > | [E D C] <~ cell[EDC]> | [F E C] <~ cell[FEC] > | [G F C] <~ cell[GFC] >

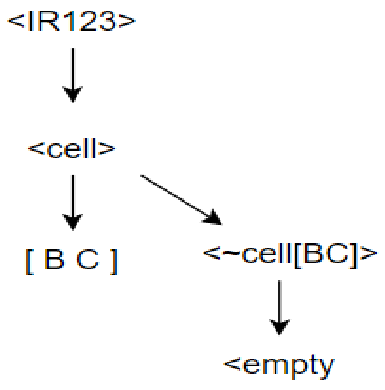
<~ cell[BC] > ::= <Empty> | [C] <~ cell[C]> | [D C] <~ cell[DC] > | [E D C] <~ cell[EDC]> | [F E C] <~ cell[FEC] > | [G F C] <~ cell[GFC] >

<~ cell[EDC]> ::= <Empty> | [C] <~ cell[C]> | [D C] <~ cell[DC] > | [B C] <~ cell[BC] > | [F E C] <~ cell[FEC] > | [G F C] <~ cell[GFC] >

<~ cell[FEC] > ::= <Empty> | [C] <~ cell[C]> | [D C] <~ cell[DC] > | [B C] <~ cell[BC] > | [E D C] <~ cell[EDC]> | [G F C] <~ cell[GFC] >

<~ cell[GFC] > ::= <Empty> | [C] <~ cell[C]> | [D C] <~ cell[DC] > | [B C] <~ cell[BC] > | [E D C] <~ cell[EDC]> | [F E C] <~ cell[FEC] > 3 – 2)

IR123 Parse Tree for: [B C]



3 – 3)

IR123 Parse Tree for: [C] [E D C] [F E C] [B C]

$$\begin{tikzcd}
\text{<IR123>} \arrow[d] & & & & \\
\text{<cell>} \arrow[d] \arrow[dr] & & & & \\
\text{[C]} & \text{<~cell[C]>} \arrow[d] \arrow[dr] & & & \\
& \text{[E D C]} & \text{<~cell[EDC]} \arrow[d] \arrow[dr] & & \\
& & \text{[F E C]} & \text{<~cell[FEC]} \arrow[d] \arrow[dr] & \\
& & & \text{[B C]} & \text{<~cell[FEC]} \arrow[d] \\
& & & & \text{<empty>}
\end{tikzcd}$$

3 – 4)

I cannot draw a parse tree consistent with my grammar for in the IR123 language for [D C][B C][B C][C] because I have no rule that allows [B C] to be placed next to another [B C]. That is the same for any cell in this language, it does not allow for adjacent repeats.

Problem 4 – BXR

4 – 1)

<BXR> ::= (<Action>) | <Boolean>

<Booleans> ::= <Boolean> <Booleans>

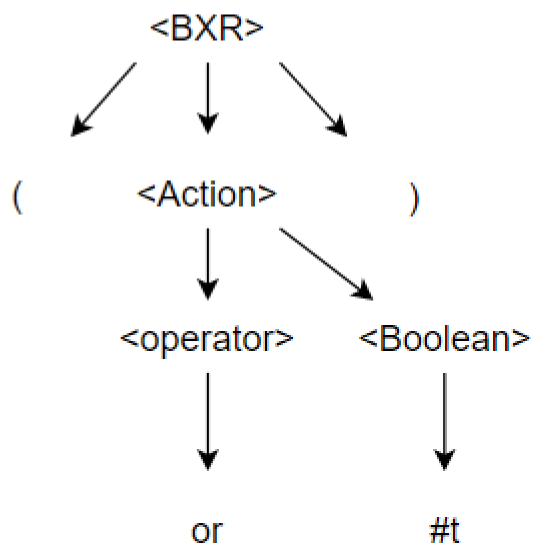
<Boolean> ::= #t | #f

<Action> ::= <empty> | <Booleans> | <operator> <booleans> | (<Action>) <Action> | (<Action>)

<operator> ::= and <Action> | not <Action> | or <Action>

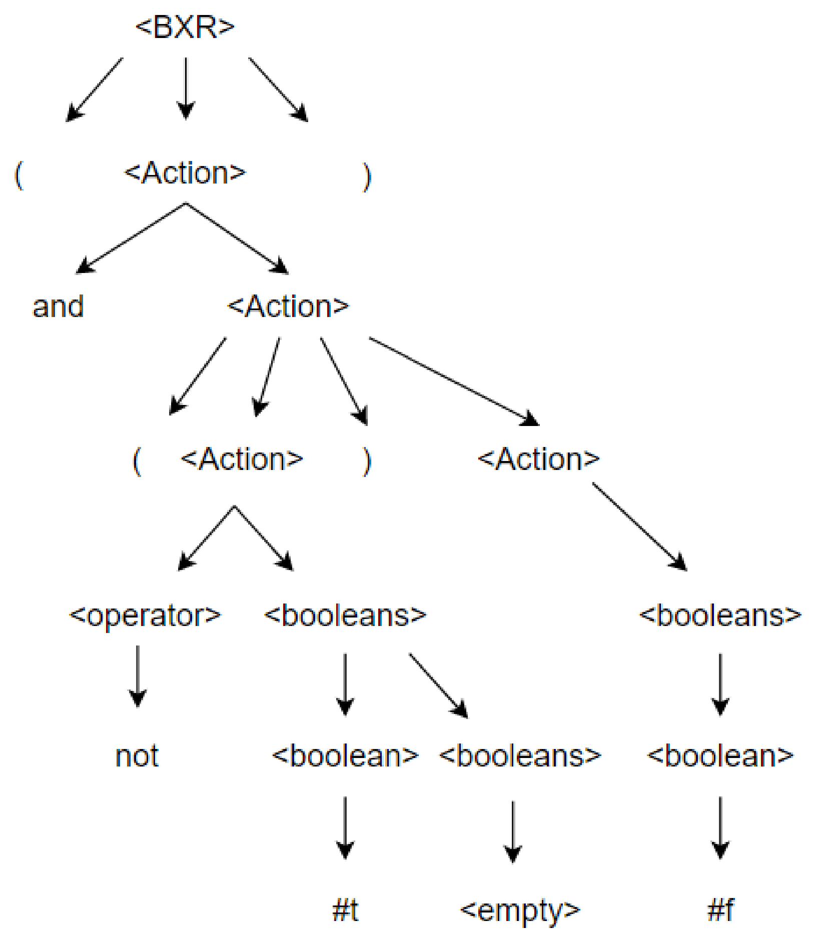
4 – 2)

BXR Parse tree for: (#t)



4 – 3)

BXR Parse tree for: (and (not #t) #f)



Problem 5 – CF (Color Fun)

5 – 1)

<cf> ::= <add> | <describe> | <colors> | <show> | <exit>

<add> ::= add (<rgb> <rgb> <rgb>) < variable-name> | add (<rgb> <rgb> <rgb> <rgb>) <variable-name> | add color < variable-name>

<variable-name> "Any string input"

<rgb> 0 | 1 | 2 | 3 | … | 254 | 255

<describe> ::= describe < variable-name>

<colors> ::= <variable-name> …. N<variable-name>

<show> ::= show <variable-name>

<exit> ::= exit

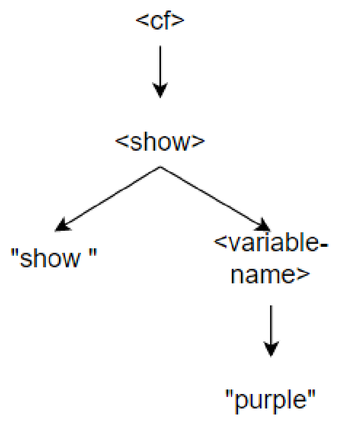
5 – 2)

CF Parse tree for: colors

$$\begin{tikzcd}
\text{<cf>} \arrow[d] \\
\text{<colors>} \arrow[d] \\
\text{<Variable-name>}
\end{tikzcd}$$

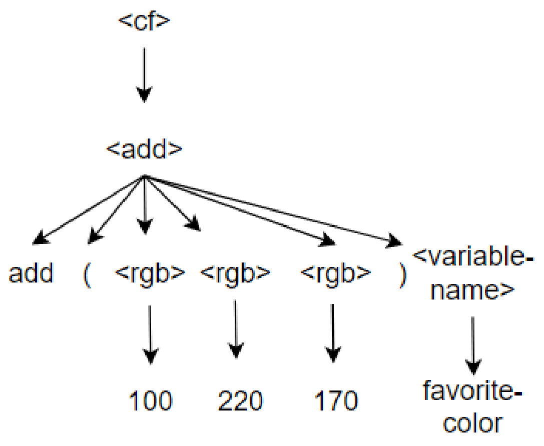
5 – 3)

CF Parse tree for: show purple



5 – 4)

CF Parse tree for: add (100 220 170) favorite-color



Problem 6 – BNF?

BNF is a model with which someone can break down and analyze any language. This is especially helpful in programming languages. By breaking down and examining languages you can improve the logic behind its structure. Some languages can do more with less grammar than others. BNF can help someone find those advantages or weaknesses in a language.